Gaussian Mixture Model Uncertainty Learning (GMMUL) Version 1.0 User Guide

Alexey Ozerov [1], Mathieu Lagrange [2] and Emmanuel Vincent [1]

[1]INRIA, Centre de Rennes - Bretagne Atlantique
Campus de Beaulieu, 35042 Rennes cedex, France
[2] IRCAM, STMS, UPMC, CNRS - UMR 9912
1 place Igor Stravinsky, 75004 Paris
{alexey.ozerov, emmanuel.vincent}@inria.fr, mathieu.lagrange@ircam.fr

January 10, 2012

1 Introduction

This user guide describes a method that can be considered for learning Gaussian Mixture Models (GMMs) while acknowledging the fact that the data can be uncertain. The level of uncertainty can be known or estimated. In order to fully grasp the theoretical aspects of the method as well as the practical facts that motivated the proposal of this method, the reader is strongly encouraged to read the following papers [1] [2] that will continuously be referred to in the remaining of this document.

This guide is organized as follows. Section 2 describes the two core functions that allows the user to learn GMMs from uncertain data. Those functions are then presented within two evaluation frameworks respectively considered in [1] and [2]. Section 3 describes the first one that focus on synthetic data, *i.e.* observed data and uncertainty is generated from sampling of GMMs. Section 4 describes the second one, that considered noisy speech data where the uncertainty is known as a prior or estimated.

2 Processing Functions

2.1 GMM EM uncertainty learning

The core function is `GMM_EM_uncertainty_learning` that implements a new Expectation-Maximization (EM) algorithm for learning GMMs from uncertain data. It can be considered for the purpose of training GMMs and decoding GMMs, both with handling of uncertainty.

```
function [uXe, cXe, wXe, log_like_N, l] = ...
    GMM_EM_uncertainty_learning(y, cE, uX, cX, wX, nbIterations, print_log_flag)

%
% [uXe, cXe, wXe, log_like_N, l] = ...
%     GMM_EM_uncertainty_learning(y, cE, uX, cX, wX, nbIterations, print_log_flag);
%
% Expectation-maximization (EM) algorithm for training Gaussian mixture
% models (GMMs) from noisy observations with Gaussian uncertainty
%
% input
% -----
%
% y                : observations (N vectors of length M), (M x N) matrix
% cE               : (opt) Gaussian uncertainty, full or diagonal covariance
%                    matrices that can be:
%                        - (M x M x N) tensor in case of full covariances
%                        - (M x N) matrix in case of diagonal covariances
%                        - empty (= []) (equivalent to conventional training)
%                    (def = [])
% uX               : initial GMM Gaussian means, (M x K) matrix
% cX               : initial GMM Gaussian full or diagonal covariances that can be:
%                        - (M x M x K) tensor in case of full covariances
%                        - (M x K) matrix in case of diagonal covariances
% wX               : initial GMM Gaussian weights, K-length vector
% nbIterations     : (opt) number of EM algorithm iterations (def = 30)
% print_log_flag   : (opt) printing log flag (def = 1)
%
%  where  K : number of gaussians in GMM
%         M : dimensionality of feature vector
%         N : number of observations
%
% output
% ------
%
% uXe              : estimated GMM Gaussian means, (M x K) matrix
% cXe              : estimated GMM Gaussian full or diagonal covariances that can be:
%                        - (M x M x K) tensor in case of full covariances
%                        - (M x K) matrix in case of diagonal covariances
%                    NOTE: cXe has the same dimentionality as its
%                    initialization cX
% wXe              : estimated GMM Gaussian weights, K-length vector
% log_like_N       : N-length array of log-likelihoods for each observation
% l                : nbIterations-length array of global log-likelihoods over
%                    EM algorithm iterations
```

Figure 1: Header of the main processing function

For decoding GMMs, one simply needs to provide the data, optionally with a Gaussian uncertainty, and the parameters of the model. In this case, the number of iteration can be set to 1, and the log-likelihood computed at the Expectation step is considered.

For training GMMs, one needs to provide the data with a Gaussian uncertainty, a first estimate of the parameters of the model and a number of iterations that is sufficient to reach convergence. In all the experiments reported in [1] and [2], those estimates are obtained with the function `VQ` discussed below.

Details about inputs and outputs are described In both cases, if the uncertainty is not provided, the algorithm reduces to a standard EM algorithm for training GMMs [3]. Detailed description of the input and output parameters is given in Figure 2.1.

2.2 GMMs initialization

The `VQ` implements a hierarchical clustering algorithm to provide a first estimate of the GMMs models. It does not consider uncertainty. Therefore, the only input parameters are the actual data

```
function [means,covars,index] = VQ(x,niveau_max,diag_covs_flag)

%
% [means,covars,index] = VQ(x,niveau_max,diag_covs_flag);
%
% An hierarchical clustering (vector quantization (VQ) or K-means)
% algorithm
%
% input
% -----
%
% x              : observations (T vectors of length n), (n x T) matrix
% niveau_max     : desired number of splits in VQ
% diag_covs_flag : (opt) estimate diagonal covariances flag (def = 1)
%
% output
% ------
%
% means          : (n x Q) matrix of cluster means
% covars         : cluster full or diagonal covariances that can be:
%                      - (n x n x Q) tensor in case of full covariances
%                        (diag_covs_flag = 1)
%                      - (n x Q) matrix in case of diagonal covariances
%                        (diag_covs_flag = 0)
% index          : (2^niveau_max+1)*1 - length vector of splits
%
%   where : Q (<= 2^niveau_max) is the resulting number of clusters
```

Figure 2: VQ function

and the number of binary splits and the type of covariance matrix to be estimated (full or diagonal covariance). After a hierarchical split of the data operated along the axes of maximal variance, the function outputs a first estimate of the GMM parameters, see Figure 2 for further details.

3 Experiment on Synthetic Data

The code discussed here has been considered to generate the results discussed in [1] with the additional case of conventional noisy training and uncertainty decoding.

3.1 Data

The data consists in synthetically generated features using GMM sampling. The uncertainty is here sampled from a signal Gaussian and supposed to be known both for the training and testing phases, see [1] for further details.

3.2 Code

The function `EXAMPLE_1__synthetic_2D_data` demonstrate the use of uncertainty learning for the purpose of classifiying synthetic data. As can be seen on Figure 3, it mainly consists in training and testing phases for each classes using the proposed approach (Uncertainty training on noisy data) and conventional ones (Conventional training on clean or noisy data).

The Figure 4 gives a sense of the results for a 2-dimensional case.

```
function EXAMPLE_1__synthetic_2D_data()

% comments and generation of synthetic data
...

for train_mode = train_modes
    %display info
    ...

    % TRAIN
    % ---------------------------
    for cl = 1:nbClasses
        fprintf('Train model %d of %d models\n', cl, nbClasses);

        x_noisy_train_cl = squeeze(x_train_CUR(cl, :, :));
        cE_train_cl = squeeze(cE_train_CUR(cl, :, :));

        [uXe, cXe] = VQ(x_noisy_train_cl, log2(nbGaussians), 0);  % 0: initialize full ↩
            covariances

        wXe = ones(1, nbGaussians);
        wXe = wXe/sum(wXe);

        [est_gmms{train_mode,cl}.u_gmm, est_gmms{train_mode,cl}.c_gmm, est_gmms{↩
            train_mode,cl}.w_gmm] = ...
            GMM_EM_uncertainty_learning(x_noisy_train_cl, cE_train_cl, uXe, cXe, wXe, ↩
                [], 0); % 0: no log
    end;


    % TEST
    % ---------------------------
    log_likelihoods = zeros(nbClasses, nbSequences_test, nbClasses);
    for cl = 1:nbClasses
        fprintf('Test for class %d of %d classes\n', cl, nbClasses);

        for seq = 1:nbSequences_test
            for cl_model = 1:nbClasses
                [dummy1, dummy2, dummy3, dummy4, log_likelihoods(cl, seq, cl_model)] = ↩
                    ...
                    GMM_EM_uncertainty_learning(squeeze(x_noisy_test(cl, seq, :, :)), ↩
                        squeeze(cE_test(cl, seq, :, :)), ...
                    est_gmms{train_mode,cl_model}.u_gmm, est_gmms{train_mode,cl_model}.↩
                        c_gmm, est_gmms{train_mode,cl_model}.w_gmm, 1, 0); % 0: no log
            end;
        end;
    end;

    % compute score
    ...
end;

% visualization
...
```

Figure 3: Code of the experiment on synthetic data.

4 Speaker recognition experiment on Speech Data

In order to experiment with a more realistic task, we considered in [2], a speaker recognition task on noisy speech data. In this case, the uncertainty is not know as a prior, and estimated using a method based on Wiener filtering and Vector Taylor Series (VTS) expansion, see Section 4.1.5 of [2] for more details.

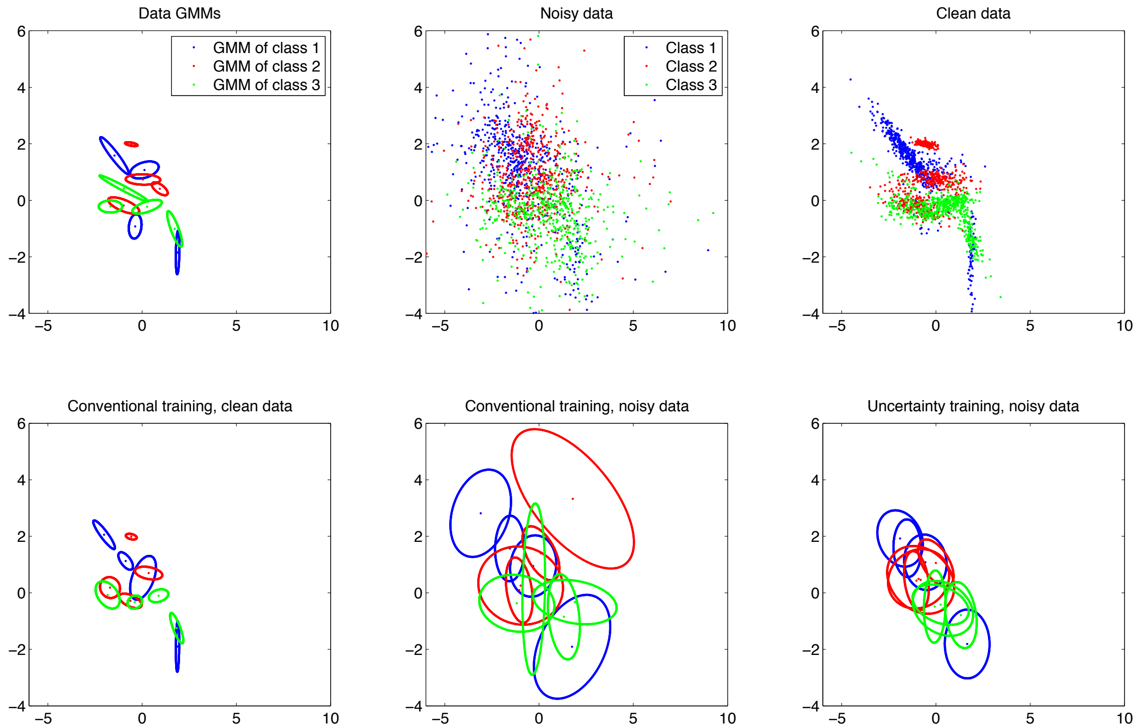

Figure 4: GMMs estimated using different learning conditions. By considering the uncertainty in the learning algorithm, the resulting GMMs are much closer to the original ones when considering noisy data.

4.1 Data

The data is provided separately from this toolbox and can be freely obtained at `http://www.irisa.fr/metiss/ozerov/Software/SP_REC_Uncrt_MFCC.zip`. It consists of Mel Frequency Cepstral Coefficients (MFCCs) computed over 3 different inputs:

1. `mix`: raw addition of clean speech and noise
2. `ssep`: output of a state of the art source separation algorithm fed with the raw addition of clean speech and noise
3. `ssep_uncrt`: same, but with an estimate of the uncertainty of the output done with the VTS method

Details about how and on what kind of audio data the MFCCs have been generated is provided in Section 4.1.2 of [2].

Data is provided as .MAT file that can be opened with Matlab® or any parser that can read HDF5 files. The naming convention is as follows: `s<speakerId>_<utteranceId>__mfcc.mat`. The speakerId corresponds to the numeric id of the speaker from 1 to 34. For the 3 different inputs described above, the .MAT file contains the `mfcc` variable which is a 2 dimensional floating point matrix of size $20 \times n_f$, where n_f is the number of frames that have been considered for computing the 20 dimensional MFCCs. For the last input, a second variable `mfcc_covar` is available. It encodes the uncertainty as a 3 dimensional tensor of size $20 \times 20 \times n_f$.

The data is divided in two main directories (test and train) that respectively contains the data used for training and testing. For each one, clean (no noise addition) and several Signal to Noise ratios are considered from -6dB to 9dB of SNRs. For each SNR, the 3 above discussed conditions (`mix`, `ssep`, `ssep_uncrt`) are available. For the latter, the full covariance uncertainty estimated using the Wiener / VTS approach is provided. It shall be noted that the MFCCs of this condition are the same as in the `ssep` condition as the VTS estimator does not change the actual MFCC values, see Equation C.2 in [2].

4.2 Code

The function `EXAMPLE_2__real_19D_MFCC_data` can be considered to replicate the full results of the experiments reported in Table D.7 of [2]. One run computes the results for one training/testing condition for the following 4 cases:

1. Conventional training / Conventional decoding without signal enhancement
2. Conventional training / Conventional decoding with signal enhancement
3. Conventional training / Uncertainty decoding with signal enhancement
4. Uncertainty training / Uncertainty decoding with signal enhancement

Figure 5 shows the first line of the function that set the main processing parameters:

- `data_dir_name`: path to the data repository
- `speaker_ids`: selection vector for the speaker to consider. $1 : 34$ will consider all the speakers available
- `subdir_name_test`: SNR condition for training from 'm6dB' (-6 dB SNR) to '9dB' (9 dB SNR), and 'clean' (∞ SNR)
- `subdir_name_train`: SNR condition for testing

On a standard 2 GHz machine with one core, the example shown on Figure 5 with 3 speakers needs about an hour. The evaluation of one training/testing condition (for example -9dB SNR at training and 0 dB SNR at testing) is expected to take one day for the 34 speakers available. So, replicating the results of the full table is expected to take about 48 days.

References

[1] A. Ozerov, M. Lagrange, and E. Vincent, "GMM-based classification from noisy features," in *Proc. 1st Int. Workshop on Machine Listening in Multisource Environments (CHiME)*, Florence, Italy, September 2011, pp. 30–35.

[2] ——, "Uncertainty-based learning of gaussian mixture models from noisy data," *Computer Speech and Language*, 2011, submitted.

[3] A. P. Dempster, N. M. Laird, and D. B. Rubin., "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, pp. 1–38, 1977.

```
function EXAMPLE_2__real_19D_MFCC_data()

% comments
....

data_dir_name = '../SP_REC_Uncrt_MFCC/';

speaker_ids = [2, 4, 6];     % can be any from 1 to 34

subdir_name_test  = 'm6dB';  % can be '0dB', '3dB', '6dB', '9dB', 'm3dB', 'm6dB'
subdir_name_train = 'm6dB';  % can be '0dB', '3dB', '6dB', '9dB', 'm3dB', 'm6dB'
```

Figure 5: Main parameters of the speaker recognition task.